Reparenting Granules Design

Goal

Provide a way to move all the granules in one collection to another collection.

Traceability

CMR-1919 - JIRA project doesn't exist or you don't have permission to view it.

Background

CMR and GCMD data is being reconciled. As this is being done providers may decide that a GCMD DIF collection is the canonical version. We will need to be able to move all of the granules from an existing collection into another new collection. This won't happen all the time and it's expected that it will be executed by a CMR administrator when requested.

Assumptions/Requirements

- The provider will stop ingesting into the source collection prior to reparenting.
- An administrator will kick off reparenting and monitor it via logging.
- We should be able to reparent multiple collections at the same time.
- We will not need to reparent granules to a collection in another provider.
    - Note the design below could support that but we'd need to update the code so that we generated a new concept id when this happens.
- Reparenting should complete quickly. A large collection should take hours not days to complete.

High Level Plan

Reparenting will be implemented by adding a new endpoint in the bootstrap application. Implementing in Bootstrap is appropriate for the following reasons:

- It is expected administrators will only need to run it.
- Bootstrap embeds Metadata DB and Indexer which allows it to retrieve, create, and index large numbers of concept revisions quickly since it avoids making network hops.

Bootstrap will asynchronously create new revisions of granules that are in the new collection.

1. Find the latest revision of all granules in a collection
2. Filter out tombstones
3. Filter out the ones that are already children of the new collection
4. Update the metadata and extra fields to point to the new collection
5. Save updated revision to metadata db and index it
6. Index the updated revision

Core Async Process

Bootstrap uses the Clojure core.async library that allows defining concurrent processing steps that communicate via channels. The following steps define each concurrent processing step and what happens.

Find batches of granule concept ids in parent collection

We'll use a work table to identify the set of granules to reparent. The reparent_work table will contain granule concept ids and the parent collection id being worked. The table can be used in parallel for different collections since the collection id provides uniqueness. The table should be cleared with the collection id when starting a run to remove rows from previous runs. The table should also be cleared after the process has completed successfully.

Populate the work table with the following sql:

```
insert into reparent_work values (select distinct concept_id, parent_collection_id from granules where
parent_collection_id = ?)
```

Select batches of granule concept ids from the table to use. Put each batch on the channel.

Find latest revisions of a batch of granule concept ids

Use existing functions that take a set of concept ids and return latest concepts. We'll process each batch of concept ids and place a batch of the retrieved latest revisions of the concepts on the next channel for processing.

Process Batch

- Filter items from each batch
    - Filter out tombstones and ones that are already children of the new collection.
    - We might find granules that have a revision in one collection and in another collection if reparenting had run once already.
- Update the metadata and extra fields to point to the new collection
- Put each individual granule on a new channel

Save and index each updated granule

We'll save and index each granule from the channel using the cmr.bootstrap.data.util/save-and-index-concept function.

Testing

We'll write integration tests similar to the other bootstrap integration tests.

When we're testing this stuff we should ingest data into one collection with multiple revisions of granules deleted and not deleted, reparent the granules, then search to find stuff, then ingest updates into the new collection. We want to make sure that the provider will see normal behavior ingesting into one collection and then ingesting the same granule into another after reparenting.

At the end of the test we should try deleting the source collection and then deleting the new collection. The granule should still exist after deleting the original collection and should be deleted after deleting the new collection.

We'll also need to do performance testing in workload.